

**TECHNICAL REPORT
NATICK/IR-04/018**



AD _____

THE V1.0 'PUSHPIN' NIXEL 2-D SELF-ASSEMBLING DISPLAY ARRAY

**by
Joseph Jacobson
Neil Gershenfeld
and
Bill Butera**

**Massachusetts Institute of Technology Media Laboratory
Cambridge, MA 02139**

August 2004

**Final Report
February 2001 – August 2003**

Approved for public release; distribution is unlimited

**Prepared for
U.S. Army Research, Development and Engineering Command
Natick Soldier Center
Natick, Massachusetts 01760-5056**

20040902 066

DISCLAIMERS

The findings contained in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of trade names in this report does not constitute an official endorsement or approval of the use of such items.

DESTRUCTION NOTICE

For Classified Documents:

Follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX.

For Unclassified/Limited Distribution Documents:

Destroy by any method that prevents disclosure of contents or reconstruction of the document.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) 24-08-2004		2. REPORT TYPE FINAL		3. DATES COVERED (From - To) FEB 01 - AUG 03	
4. TITLE AND SUBTITLE THE V1.0 'PUSHPIN' NIXEL 2-D SELF-ASSEMBLING DISPLAY ARRAY				5a. CONTRACT NUMBER DARPA BAA DAAD16-00-R-0012	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Joseph Jacobson, Neil Gershenfeld and Bill Butera				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology Media Laboratory 20 Ames Street, Cambridge, MA 02139				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Army Research, Development and Engineering Command Natick Soldier Center ATTN: AMSRD-NSC-TP-S 1 Kansas Street Natick, MA 01760-5056				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) NATICK/TR-04/018	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This report details the results of the MIT - RF Nixel Program. The RF Nixel Program represents a novel architecture for display systems based on an ensemble of Nixel display elements. Each Nixel element comprises a display element (pixel), lightweight computation sufficient to run self-assembling software code elements and local communications. A proof-of-principle hardware implementation (The V1.0 'Pushpin' Nixel 2D array) was carried out and used to successfully run the gradient and coordinate generating algorithms. Detailed simulation shows the viability of such an approach to render complex text and graphics.					
15. SUBJECT TERMS <div style="display: flex; justify-content: space-between;"> <div style="width: 30%;"> FLEXIBLE DISPLAYS DISPLAY SYSTEMS COMPUTER ARCHITECTURE PIXELS </div> <div style="width: 30%;"> ALGORITHMS CODING SELF ASSEMBLING TEXT PROCESSING </div> <div style="width: 30%;"> GRAPHICS COMPUTER PROGRAMS COMPUTERIZED SIMULATION NIXELS </div> </div>					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES 26
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	19a. NAME OF RESPONSIBLE PERSON Henry Girolamo		19b. TELEPHONE NUMBER (Include area code) 508-233-5483

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES	iv
PREFACE	v
SUMMARY	1
1. INTRODUCTION AND BACKGROUND	2
2. TECHNICAL APPROACH	2
3. THE NIXEL DISPLAY	3
3.1 HARDWARE ARCHITECTURE	3
3.2 PROGRAMMING MODEL	6
3.3 SIMULATION TOOLS	7
4. TEXT DISPLAY	8
4.1 ANCHORS AND GRADIENTS	8
4.2 2-D COORDINATES	10
4.3 TEXT RENDERING	11
5. GRAPHICS CONTROL	13
6. RELATED WORK	15
7. CONCLUSIONS AND RECOMMENDATIONS	16
REFERENCES	18

LIST OF FIGURES

Figure 1. Process and Data Flow in Display Systems	3
Figure 2. The V1.0 'Pushpin' Nixel	4
Figure 3. Programming Model	7
Figure 4. Gradient – Run Time View	9
Figure 5. Gradient - Adaptation Procedure	10
Figure 6. Adaptation Rules for Gradient pfrag	11
Figure 7. Coordinate System: Construction and Adaptation	11
Figure 8. Text Seed: Positioning and Rendering	12
Figure 9. Text pfrag: Post Definition and Messaging Topology	13
Figure 10. Text pfrag: Adaptation Rules for Spawned Copies.	14
Figure 11. Translation and Scaling	14
Figure 12. A 2-D Array of V1.0 'Pushpin' Nixels Running a Gradient Algorithm.	17

LIST OF TABLES

Table 1. The V1.0 'Pushpin' Nixel Parts List	5
--	---

PREFACE

The RF Nixel V1.0 Pushpin Hardware Simulator and Software were developed by the Massachusetts Institute of Technology Media Laboratory (MIT Media Lab) under the program management of the DARPA-MTO Office, Contract Number DARPA BAA DAAD16-00-R-0012, (Robert Tulis, PM). The interim program was completed under the direction of the US Army Natick Research, Development and Engineering Command from February 2001 to August 2003. The purpose of this program was to develop a hardware simulator and software to demonstrate the feasibility of a new type of distributed display based on self-assembling code. This is the final report for the interim RF Nixel Program.

THE V1.0 'PUSHPIN' NIXEL 2-D SELF ASSEMBLING DISPLAY ARRAY

SUMMARY

An alternate architecture for display systems is reported based on an ensemble of Nixel display elements running self-assembling software code elements. Each Nixel element comprises a display element (pixel), lightweight computation sufficient to run such self-assembling software code elements and local communications. A proof-of-principle hardware implementation (The V1.0 'Pushpin' Nixel 2-D array) was carried out and used to successfully run gradient and coordinate generating algorithms. Detailed simulation shows the viability of such an approach to render complex text and graphics.

1. Introduction and Background

Consider the scaling limitation for large area, real time display systems. For systems based on current display architectures (Figure 1), data from a variety of external image sources (both natural and synthetic) must first pass through a graphics engine for rendering and formatting. The rendered output is then channeled to a display component for distribution to individually addressable display elements (pixels). Efforts to scale up the number of pixels ultimately confront a number of obstacles, both quantifiable engineering hurdles and more subtle impediments to acceptance by the end users. Engineering limitations are typically caused by shared resources that bottleneck; *e.g.* graphics engines with finite aggregate compute capacity, and data buses with bounded transmission bandwidth. Manufacturing processes for the displays likewise have difficulty maintaining adequate yield as the number of pixels per display increases. For the end customer, ultrahigh resolution wide area displays must be treated as fixed objects that are mechanically and electrically sensitive, require substantial infrastructure, are often bulky, are always complex, and are difficult to reconfigure opportunistically. The last decade has seen dramatic headway made on a portion of this problem space – namely that of the display component. Emblematic of this is the work on electrophoretic ink (*e-ink*) where advances in the production and handling of microcapsules have yielded bistable, printable displays with the viewing affordances of paper and the dynamic display updating of a CRT [3] [6].

Nevertheless, progress on the display component has only underscored the unmet challenges at the system level. Design of back end systems with sufficient rendering power and transmission capacity to feed a 10^9 pixel display in real time still outpaces today's best engineering practice. Contemporary solutions adopt the approach of tiling the active display area among autonomous display systems operating in synchrony, much in the spirit of early systems that rack mounted multiple TV's into a rectangular grid. This program and report looks at carrying this simple approach to a novel extreme. Specifically, *we propose assigning a full-featured graphics system to every pixel in the system.*

2. Technical Approach

Core to this approach is the architectural work on "paintable computing" and its associated programming methodology based on informational self-assembly [2]. Architecturally, we define a paintable computer as an agglomerate of numerous, finely dispersed, ultra-miniaturized computing particles; each positioned randomly, running asynchronously and communicating locally. Individual computing nodes are vanishingly cheap, freely expendable, and consequently are handled in a bulk fashion. In this report we present initial work on applying the architecture and programming model of *paint* to the design of display systems. In doing so, we motivate the question "can the display systems themselves become paintable?" The goal is to create display systems with the same characteristics we target for *paint*; scalability, ability to reconfigure, resilience to fault, and self-repair. We entitle such displays Nixel displays.

Section 3 lays out the Nixel display concept based on the underlying paintable computing ideas above. The section details both the hardware architecture as well as the

programming model for self-assembly of mobile code. Section 4 lays out the mechanics of a specific display task, that of text rendering. In this approach self-replicating process fragments ("pfrags") interact to construct a 2-D coordinate system.

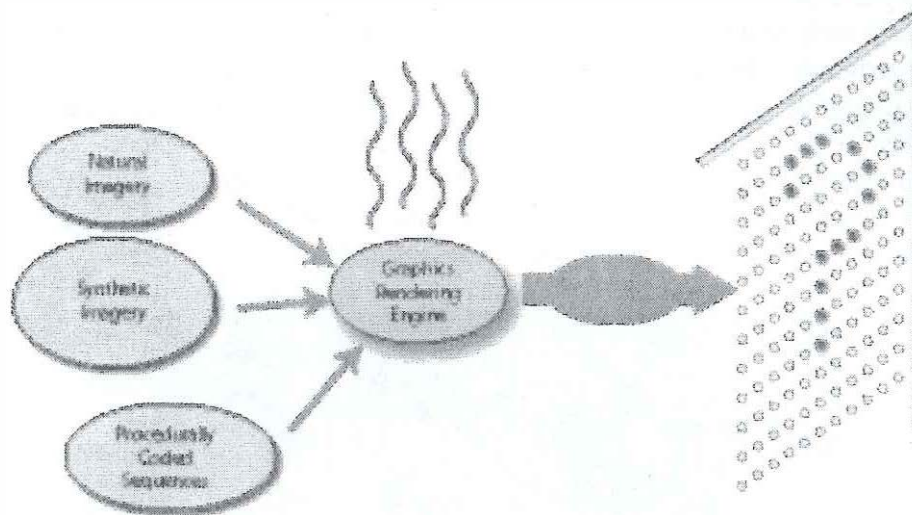


Figure 1. Process and Data Flow in Display Systems

Image data from multiple sources converges on a single graphics engine for decompression, color space conversion, rendering, and formatting. Graphics engine passes rasterized data through high bandwidth channel to display the 2-D scaffold to migrate to a predetermined position. Section 4 extends basic pfrag rendering behavior to support simple graphics control functions such as translation, scaling, and rotation. Section 5 samples related work. Section 6 states the conclusions, previews future work and expands on important system-design question raised in this report.

3. The Nixel Display

The Nixel Display Concept consists of a machine consisting of thousands of sand-grain sized processing nodes, each fitted with a display element and modest amounts of processing and memory, positioned randomly and communicating locally. Individually, the nodes are resource poor, vanishingly cheap and necessarily treated as freely expendable. Yet, when mixed together they cooperate to form a machine whose aggregate compute capacity grows with the addition of more nodes. The ultimate goal is to recast computing as a particulate additive to ordinary materials such as building materials or paint.

3.1 Hardware Architecture

The atomic element of a *paintable display* is the particle in this version entitled the V1.0 pushpin. Characteristic specs include a '486 class micro, an internal clock running at ~ 100 MHz, and 250K-1M of RAM for code and data storage. All the I/O to

the micro is gated through a wireless transceiver supporting a minimum full duplex rate of 1 Mb/s. Communication is via asynchronous links to the nearest neighbors. A power subsystem harvests power from the immediate environment with minimal constraints on the particle's placement. Once exposed to power, each particle builds an enumerated list of the neighbors with which it can communicate. The characteristic specs were enshrined into both a hardware reference model and a device simulator. In the interim, an initial COTS (1.Commercial Off-The-Shelf – a system designer's term of endearment to denote systems constructed exclusively from commercially available components-i.e. no custom parts) version of the hardware (The V1.0 'Pushpin' Nixel) is complete in small quantities (Fig. 2) with deployment/coding on an initial ensemble of 1000 proceeding. Table 1 gives the specifications for each part of the V1.0 'Pushpin' Nixel.

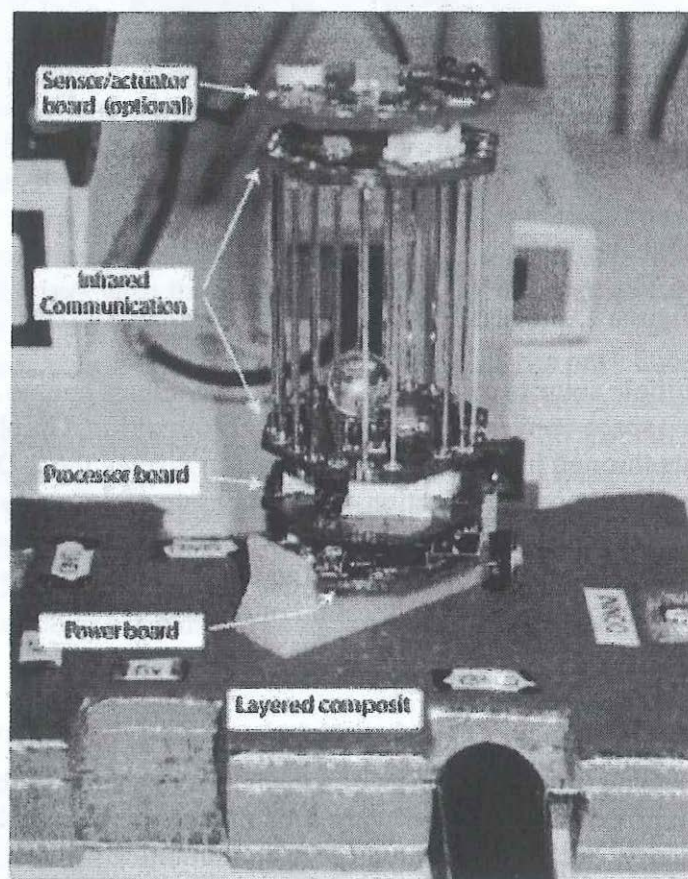
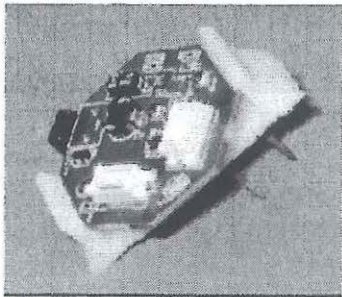
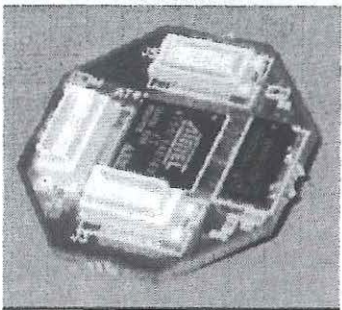
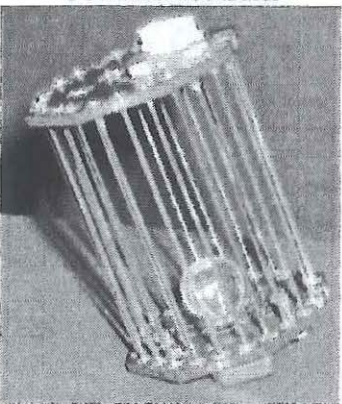
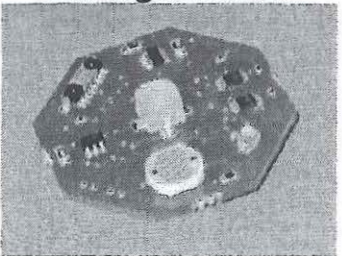


Figure 2. The V1.0 'Pushpin' Nixel

First pass COTS version of a *paint* particle. Arranged in a 5-board stack, the 1.25" x 3" form factor is comparable to that of a saltshaker. Base board (power board) fitted with catheterized nails that thumbtack into a layered composite consisting of three conducting planes separated by nonconducting planes. System specs: 32-bit RISC processor, 66 MHz clock, 256 KB SRAM and 2 MB FLASH. Communication via 4 Mbs infrared link to neighbors within a 4.5" radius.

Table 1. The V1.0 'Pushpin' Nixel Parts List

V1.0 'Pushpin'	
<p>Power</p> 	<ul style="list-style-type: none"> • 3.3v regulation • 5v regulation • Drive for □omm./debug plane • JTAG connector • Status LED (debug) • Reset circuitry
<p>Processor</p> 	<ul style="list-style-type: none"> • 32-bit Microprocessor, RAM, FLASH, digital peripherals • 1.8v supply • 66 Mhz Oscillator
<p>Communications</p> 	<ul style="list-style-type: none"> • 4 Mbs IrDA CODEC • Crystal + supporting discretes • Analog transceiver • Ir photo emitter diode • Ir photo detector • Acrylic ball (poor man's Ir antenna) • 'birdcage' style enclosure to support isotropic 4 inch communication radius
<p>Sensing/Actuation</p> 	<ul style="list-style-type: none"> • Interface to processor: 6 configurable lines – (3 for USART, 3 for Timer) • Tri-color LED for pixel display • Sensor: visible light • Sensor: proximity switch • Sensor: temperature switch

3.2 Programming Model

The programming model is based on the concept of process self-assembly -- the unsupervised re-assembly of a running process from fragments of code that are mobile in a virtual environment. Process self-assembly is loosely modeled on the metaphor of reversible self-assembly in the material world, in material self-assembly, local chaotic interactions between autonomous physical elements (biological cells, gas molecules, Wall street traders) produce global behavior that is well ordered. In process self-assembly, the atomic elements are virtual -- autonomous, mobile fragments of code with state. These process fragments (pfrags) use local messaging to emulate the forces that direct material self-assembly. They in turn use these simulated forces to direct their further migration, ultimately arriving at a predetermined spatial ordering.

Fig. 3 illustrates the two cornerstones of this programming model; pfrags that are self-contained, and memory that is locally shared and probabilistic. The shared memory model is patterned after bulletin board systems that communicate via lossy update channels. Every particle contains a local entry to the bulletin board (the Homepage) where resident executables can read and write tagged data (posts). An additional segment of memory (the I/O space) is reserved for mirrored instances of the Homepages from the neighboring particles. Posts to a given Homepage appear -- with an unbounded latency and non-zero probability of failure -- at the mirror sites on all the particles within the network neighborhood. For an executable running in a given particle, the Homepage is read-write and the I/O space is read-only.

All software executed in *paint* is represented as process fragments. Pfrags are self contained and sized to fit entirely in the RAM space of a single particle 1. Inter-pfrag messaging is coded as posts 2 and gated through the I/O space. All writes must be to the Homepage. Reads can come from anywhere in the I/O space. In response to periodic interrupts from the particle's OS, a pfrag will scan the I/O space and, depending on what it finds there, will execute one of several predefined behaviors -- post more data on the Homepage, request a transfer to a neighboring particle, self-delete, or simply idle. 1. none the pfrags of this report are larger than 600 bytes in size. 2. 'posts' are defined as variable length key-value pairs.

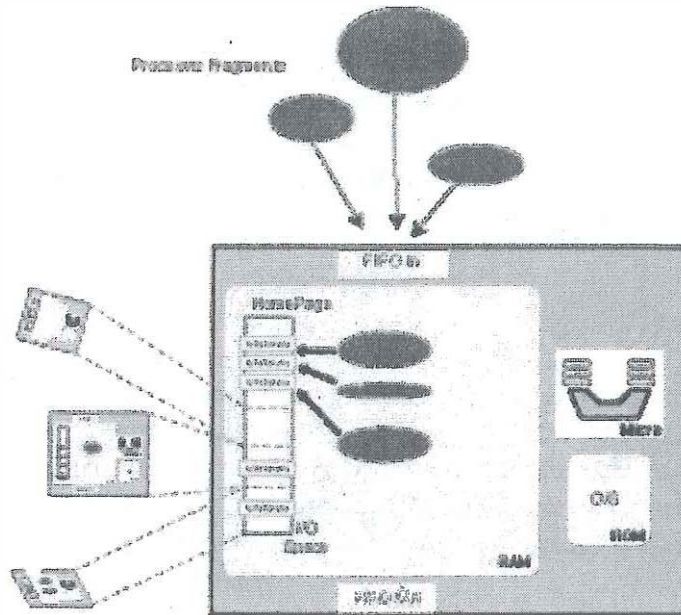


Figure 3. Programming Model

Distributed processes are embodied as ensembles of autonomous migratory process fragments ("pfrags"). Inter-pfrag communication via tagged messages posted on local bulletin board memory segment (Homepages). These messages are referred herein as **posts**. Copies of the Homepages from neighboring particles assembled into read-only I/O space.

The pfrag's continual, free running evaluation of its environment as encoded in the I/O space is the key driver for adaptation on *Paint*. For example, if a pfrag read from the I/O space the value of the variables *y* and *z* and bound their sum to an internal variable *x*, then the pfrag would constantly be re-adding *y* and *z* to keep the value of *x* current. Section 3 expands on the utility of this approach.

3.3 Simulation Tools

The work reported here predates the availability of the pushpin hardware, and was consequently developed and tested on the device simulator. Written in Java 1.1, this simulator models each particle functionally as an instance of a template Java object. An associated viewer supports visualization, and generation of image sequences. A GUI allows execution control (Run/Step/Stop), and selection of salient system parameters such as communication radius, number of particles, degree of randomness in placement, and placement of I/O portals for insertion of pfrags into ensemble.

4. Text Display

The method for positioning and display of a text character is patterned after the behavior of a "directed seed". Seeds are single pfrags containing a character modeled as a sequence of geometric primitives, and a rendering algorithm capable of interpreting the model to compute an RGB pixel value. As a precursor to "planting the seed", two reference points are externally declared and serve as insertion points for a sequence of pfrags that interact to build a 2-D coordinate system. With this scaffold in place, the text pfrag is then inserted, migrates to its pre-assigned position, and covers a bounded neighborhood with lightweight copies of itself. These copies act as portable rendering programs that are customized for the rendering of the one single character.

The remainder of this section describes this method in detail, with special attention on the adaptation strategies employed by individual pfrag types. Sample results illustrate how individual pfrag adaptation, together with local messaging, enable groups of pfrags to cooperatively form robust informational structures.

4.1 Anchors and Gradients

Creation of a 2-D coordinate system begins with the selection of two externally defined reference points and the establishment of a gradient field centered about each of these points. Gradient fields as a natural phenomena (thermal, chemical, electromagnetic) have become a common abstraction for messaging in physically distributed systems, with individual communities developing their own domain-specific variation. In the paintable PM, the generic gradient is realized as a lightweight pfrag that enters the particle ensemble through a single point and propagates virally, ultimately inserting a single copy of itself into every particle. Once a region is blanketed, the gradient pfrags message locally to generate a fractional estimate of the shortest path distance back to the source. 2. Messaging between spatially proximal Gradients is mediated by posts to the Homepage (Fig. 4). Each Gradient continually scans the I/O space for posts from neighboring Gradients. In response to what it finds, a Gradient will either update its own post, delete itself, or simply do nothing. (Fig. 5) specifies this adaptation procedure.

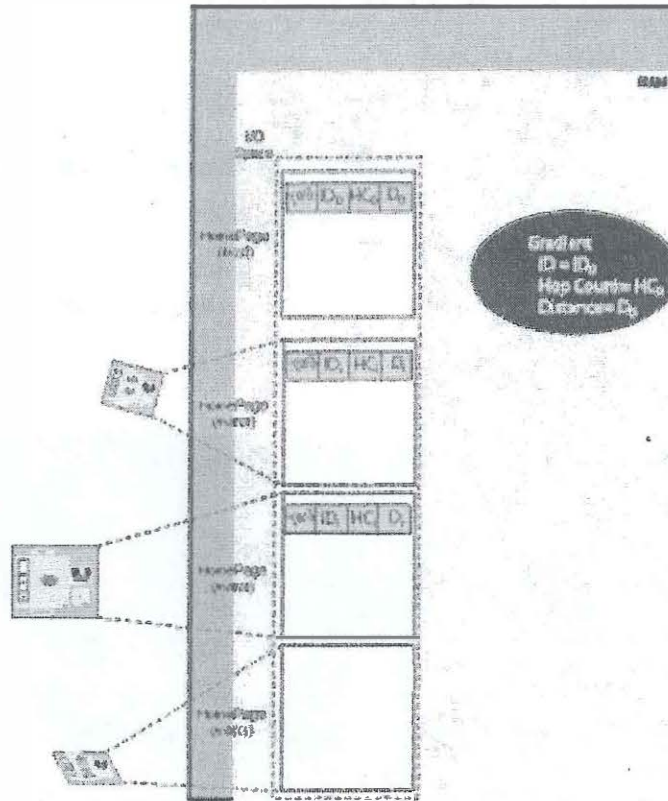


Figure 4. Gradient - Run Time view

Run time environment as viewed through the I/O space. Host particles contains only one pfrag - the Gradient. I/O space consists of local Homepage plus mirrored copies of Homepages from three neighboring particles. Gradient sees its own post in local Homepage, and posts from other Gradients in two of the three neighbors. Entries in Gradient post: Tag = key unique to Gradient pfrags ID = identifier used to group Gradients. HC = integer hop count D = fractional distance. Notes: 1. Placement of the reference points along a boundary of the particle ensemble eliminates the need for a symmetry-breaking third point 2. These distance estimates are normalized to the communication radius of the particles.

The adaptation procedure is based on two values:

- HC – the Gradient's own posted hop count
- HC_{min} – the minimum hop count found in the I/O space (i.e. The hop counts from the neighboring Gradients)

```

1. if  $HC < HC_{min} + 1$ ,
   then self-delete1
2. else if a page in the I/O space has no post from a
   Gradient,
   then propagate a copy of the Gradient to the neigh-
   boring particle.
3. else if  $HC > HC_{min} + 1$ ,
   then update post to set  $HC = HC_{min} + 1$ 
4. else,
   do nothing

```

1. Note that the point at which a Gradient is inserted into the particle ensemble is the single instance of a $HC=0$. If that Gradient is removed, all the Gradients with $HC=1$ will self-delete. The ensuing domino effect voids the particle ensemble of that pfrag until another Gradient with $HC=0$ is inserted.

Figure 5. Gradient – Adaptation Procedure

At the end of every round of adaptation, the Gradient recomputes a fractional estimate of distance by averaging over the entire neighboring integer hop counts. It then includes this fractional value in its Homepage post. This perpetual, free running re-evaluation of the environment as represented by the I/O space is core to the *paintable* approach to adaptation and self-repair.

4.2 2-D Coordinates

The arrival at one anchor point of the Gradient from the other anchor triggers the insertion of a Coordinate pfrag. Coordinate virally deposits a single copy of itself into every particle. Coordinate uses the two Gradient distance estimates to synthesize a 2D coordinate that it then includes in its Homepage post (Fig. 7a). These coordinates are normalized to the radius of the network link that defines a particle's neighborhood. The error bound on the coordinate depends directly on the accuracy of the Gradient distance estimate. This error performance has been treated both analytically and experimentally in the context of amorphous computing [9]. As with the Gradient pfrag, adaptation in the Coordinate pfrag is a by-product of the free running generative process (Fig. 6). Fig. 7 B shows the response of this process to a real failure. The particles within an irregularly shaped patch were disabled. The 2-D distortion in the immediate vicinity of the failure is due to the fact that the Gradient's shortest-path distance estimates to the corresponding anchor point no longer map to Euclidean distance. However the normalized amplitude of this distortion falls off with increasing distance from the failure zone. Self-repair follows if the area is repopulated with new particles.


```

1.if either Gradient is missing,
   then self-delete.
2.else if an uninfected neighbor is found,
   then propagate a copy to that neighbor.
3.else,
   read the Gradient posts, re-triangulate and update the
   local Coordinate post as necessary.

```

Figure 6. Adaptation Rules for Coordinate pfrag

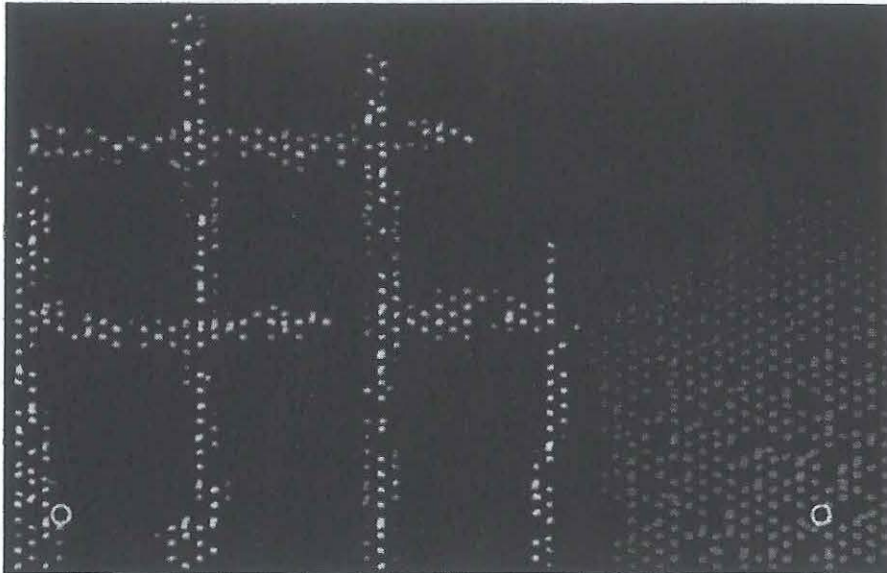


Figure 7. Coordinate System: Construction & Adaptation

Initial state is 3000-particle ensemble devoid of pfrags. In the construction phase (a), two anchor points are externally positioned (bottom) and assigned a 2-D position. Gradient fields, emanating from each anchor, append the assigned 2-D position to their Homepage posts. The arrival of a Gradient from one anchor at the other anchor triggers the insertion of a Coordinate pfrag that uses the posted Gradient distances to estimate a 2-D position. Estimation procedure is free running, and automatically adapts to subsequent particle failure. Distortion caused by an areal failure (darkened area near center) is localized (b).

4.3 Text Rendering

Individual text characters are embodied as single pfrags. A Text pfrag contains three elements in its state vector:

1. A model of the character expressed as geometric primitives such as lines, arcs and splines.
2. A preferred position on the 2-D coordinate system (relative to the 2-D coordinate system).

3. A bounding box (likewise relative to the 2-D coordinate system) A text pfrag enters the particle ensemble at a random point, migrates to its preferred 2-D location (Fig. 8 a), and then spawns lightweight copied of itself. The spawned copies limit their propagation to the spatial patch defined by the bounding box. The copies then compare the local 2-D coordinate to the character model, compute the corresponding RGB pixel value, and post this value in the HomePage1. Fig. 8b illustrates this approach applied to a single character; an upper case 'A'. The 'A' is encoded as three straight lines, each defined by two endpoints. The rendering procedure calls for:

- reading the local 2-D coordinate,
- calculating the shortest path distance to the nearest geometric primitive (a stroke)
- computing a monochrome pixel value by inversely weighting a saturated intensity, by the distance from the nearest stroke.

Note how this use of intermediary gray levels computed as a function of distance from a stroke, produces the anti-aliasing known to be important for legibility [4].

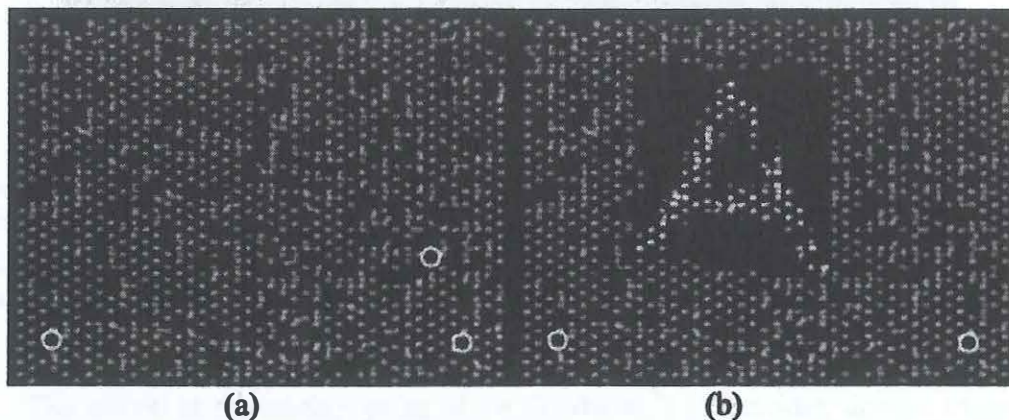


Figure 8. Text Seed: Positioning and Rendering

An ensemble of 1000 particles is initially configured with a coordinate system (not shown) relative to two anchor points (shown as I/O ports along bottom edge). A single Text seed (red) enters through a third I/O port and migrates toward a coordinate that has been pre-stored in its internal state vector.(a)(b) Once in position, the original seed (red) spawns lightweight copies of itself. The copies limit their propagation to the area defined by their internally specified bounding box. For each particle within the bounding box, the seed's copy compares the local coordinate against the geometric model of the character to arrive at an anti-aliased pixel value.

Two crucial components of this approach are adaptation, and inter-pfrag messaging between the Text seed and its copies. In order to message to its spawned copies, the Text pfrag employs a broadcast technique similar to that of the Gradient. Posts from the Text pfrags contain an integer hop count and a variable length payload appended to the end of the post (Fig. 9b). The seed Text pfrag assumes a hop count that is fixed to

zero. Text copies adopt a hop count that reflects their distance from the seed (Fig 9a). The Text seed signals to its copies by posting a new payload containing an incremented revision number. Copies seeing a recent payload in a post from a neighbor with a smaller hop count, read this payload and append it to their own post – effectively propagating the change. Beyond a revision number, payloads can contain any amount of data in any prearranged format. However in this work the payloads always contain new state vector data.

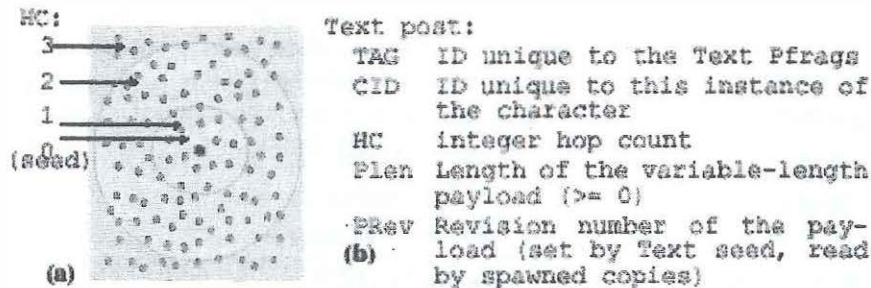


Figure 9. Text pfrag: Post Definition and Messaging Topology

The second pillar of this approach is an asynchronous, free running adaptation. The Text pfrag's instantaneous behavior is defined by its adaptation procedure operating on the static state vector and the dynamic environment. The procedure as described in Fig. 10 supports basic operations such as erasure of the character, in addition to the graphics transformations of the next section.

5. Graphics Control

Simple graphics control is recast as message passing and manipulation of the adaptation rules. With changes in the state vector automatically reflected in the rendered character, graphics manipulation can be effected by directed changes of the state vector. In our sample embodiment, an external agency broadcasts a coded graphics command 2 to the Text seeds. Each seed then parses the message, encodes the state changes into a payload, and updates its posts to include the new payload. The payload eventually arrives at the copies, which in turn internalize the new state data. The adaptation rules do the rest.

Fig. 11 illustrates this process for translation and scaling. Translation is expressed as a 2-D offset to a Text pfrag's preferred position. The seed moves, and messages the new center position to its copies. Those copies that find themselves outside the new bounding box self-delete. Other copies suddenly find unpopulated neighboring particles that are within the new bounding box, and respond by propagating new copies to

The adaptation rules for the spawned child pfrag are different than those for the primary seed.

1. if no Coordinates posts are found on the local HomePage,
then self-delete.
2. else if a neighboring particle is inside the bounding box but does not contain a post from a copy pfrag,
then propagate a copy to the uninfected neighbor.
3. else if a neighbor with a posted hop count smaller than the local hop count has a higher payload index,
then copy the new payload and update post.
4. else if the updated payload changes the bounding box and the pfrag is outside the new bounding box,
then self-delete.
5. else
recompute and repost the RGB value.

Figure 10. Text pfrag: Adaptation Rules for Spawned Copies

the unpopulated neighbors. All copies recomputed the local pixel color. Translational offsets above some threshold have the effect of sliding the rendered character out of view (Fig. 11a). However, even in cases where the copies are completely absent, the seed remains waiting to restore the character should further translation again bring the character into view.

Scaling can likewise be affected by a change in the bounding box. Fig. 11b illustrates the effect of scaling the bounding box by a factor of 0.95. The simple vehicle of resizing the bounding box suffices only within a bounded range of scale factors. As the scale factor continues to shrink, the copies will be able to apply rendering schemes of near arbitrary sophistication to maintain legibility.

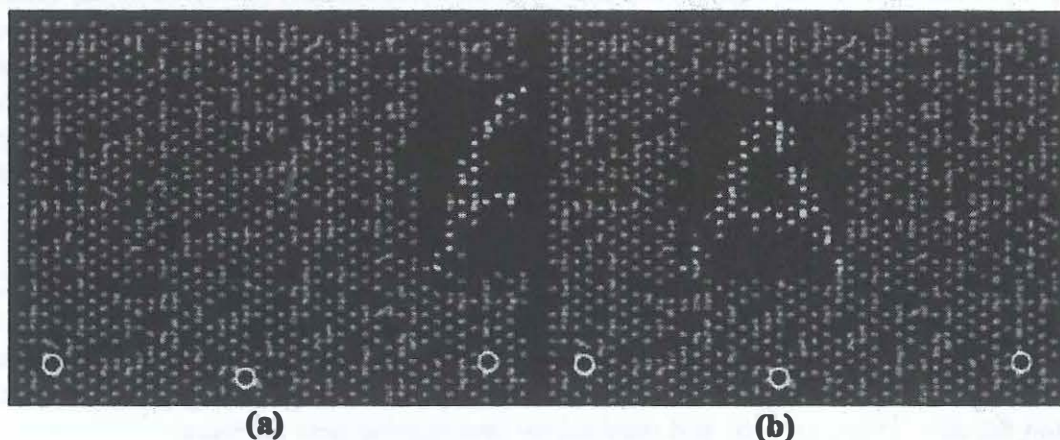


Figure 11. Translation and Scaling

6. Related Work

The vision of deeply embedded, finely grained, freely configurable computing is one that is shared among multiple communities. One such community proceeds from the assumption that relentless cost drivers in the established silicon IC sector will deliver much of the desired hardware functionality in the near term. In addition to Nixels, this guild includes groups working on wireless sensor networks [11], and those working on denser 'amorphous' computing meshes programmed around biological metaphors. At the level of node architecture, Nixels have the strongest commonality with the evershrinking sensornet nodes [7]. However the two camps diverge in their approach to software – largely due to the difference in target inter-node spacing (meters vs. millimeters) and the attendant application domains. The distributed software architecture based on process self-assembly is but one sample of a broader trend toward modeling distributed software systems on the metaphor of self-assembly as it occurs in the material world. A closely related effort in amorphous computing drew inspiration from biology to develop a programming model based on programmed self-assembly [1] and demonstrated its utility over a wide range of pattern formation; both 2-D and 3-D.

6. Conclusions and Recommendations

Much work remains and the results to date are only suggestive. Yet, to the degree that the challenges of Nixel displays yield to continuing research the work outlined in this report demonstrates that such an approach may have a significant impact in overcoming the important limitations of existing displays. Revisiting Fig. 1, the fine grain spatial distribution of the graphics function in the Nixel display approach removes the two bandwidth bottlenecks associated with channeling all input into a single point for serial processing, followed by a redistribution to the pixel raster for display. Nixel systems, where the nodes are truly autonomous, where there are no shared resources and where the inter-node messaging is strictly local, all scale freely – at least until more subtle bounds take hold. Self-repair and the ability to freely reconfigure likewise follow from modularized software built around the requirements of spatial locality and ongoing adaptation.

It is worth calculating the cost of manufacturing of a Nixel display. Assuming \$16/sq. in. for an 8-inch silicon wafer, near perfect yield, and particle die size of 0.25 mm, an ensemble of 10^6 particles (10^6 pixels) would cost ~\$6K, not wholly out of line with current costs of plasma displays for instance but allowing a completely flexible display disposed on a nearly arbitrary surface.

A compliment to lowering the system cost is to justify the system cost. This suggests a strategy of broadening the application domain by driving the development of a “Nixel display” toward subsumption of additional components from the standard computing architecture. The pixel display that already annexed the graphics controller should also incorporate the processor, the memory, the external networking, and the remaining I/O modalities³ into a single distributed machine.

What can this distributed machine do? Work to date has demonstrated its capacity for storage, communication and signal processing in the context of plausible applications [2]. Ongoing work is building on these basics to realize distributed estimation and control. In those applications where the underlying Nixel is performing some useful, compute-intensive task, addition of display becomes an incremental cost.

One of the most demanding open points relating to Nixel displays involves the use of increasingly sophisticated image models for both decoding and encoding, and the need to revisit basic hardware assumptions as the particle ensemble scales up to 10^3 or more Nixels. The Nixel architecture naturally favors compute load over transfer bandwidth. At least initially, this biases Nixels toward display of model-based coded imagery. The nuance in this case being that the models should themselves be amenable to a distributed representation.

The text characters of this report were coded using a pen stroke as an image model. The quest for ever richer display output will drive the creation of distributed versions of ever more complex models; distributed postscript for imagery coded procedurally as a sequence of pen strokes, motion-compensated frequency transformations for natural imagery coded a’la MPEG-1 and MPEG-2, a distributed variant of OpenGL for synthetic imagery, and ultimately, a distributed hybrid in the vain of MPEG-4 and MPEG-7. The dual to model-based decoding is model-based encoding. As a massive nonlinear search through a collection of heterogeneous models, model-based encoding is an interesting problem in its own right. The relevance to the Nixel

display problem is that coded imagery is only as rich and/or efficient as the models that the images are coded against. At some point, the models employed for display on Nixel hardware will necessarily take on characteristics unique to that distributed environment. We will probably have to at least visit issues relating to; coding of arbitrary pen strokes into a distributed postscript, parallel variants of hybrid waveform coder for natural images, and synthesis of open GL-like descriptions of natural objects and scenes.

Finally, in addition to software, a significant push is anticipated to evolve the hardware. Both the simulator and the pushpin environments were characterized by $\sim 10^2$ Nixels. Figure 12 below shows a gradient algorithm successfully running on \sim fifty V1.0 Nixels. As discussed above such an algorithm encompasses many of the essential features for a fully scalable Nixel display and represents solid proof of principal for the scalability of the Nixel system. With the milestone results represented in this report we eagerly look forward to scaling the Nixel system to 10^3 nixels and beyond.

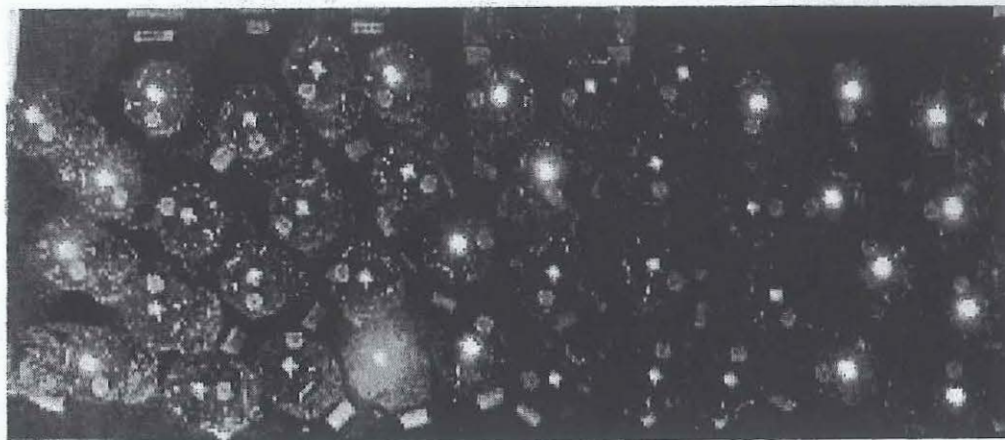


Figure 12. A 2-D Array of V1.0 'Pushpin' Nixels Running a Gradient Algorithm.

This document reports research undertaken at the U.S. Army Research, Development and Engineering Command, Natick Soldier Center, Natick, MA, and has been assigned No. NATICK/TR-04/018 in a series of reports approved for publication.

References

- [1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss *Amorphous Computing* Communications of the ACM, vol. 43, pp. 74-82, 2000.
- [2] W.J. Butera, *Programming a Paintable Computer*, Ph.D. in Media Arts and Sciences. Cambridge, MA: Massachusetts Institute of Technology, 2002, pp. 1-176.
- [3] B. Comiskey, J. D. Albert, H. Yoshizawa, and J. Jacobson, *An electrophoretic ink for allprinted reflective displays*, Nature, vol. 394, pp. 253-255, 1998.
- [4] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, Reading, Massachusetts: Addison-Wesley, 1991. pp.132-142
- [5] L. Girod, V. Bychkobskiy, J. Elson, and D. Estrin. *Locating tiny sensors in time and space: A case study*. In Proceedings of the International Conference of Computer Design (ICCD) 2002.
- [6] J. Jacobson, B. Comiskey, B. Turner, J. Albert, and P. Tsao, *The last book*, IBM Sytems Journal, vol. 36, pp. 457-463, 1997.
- [7] J. L. Hill, *System Architecture for Wireless Sensor Networks*, Ph.D. in Computer Science. Berkeley CA: University of California, Berkeley, 2003, pp. 1-186.
- [8] K. Langendoen and N. Reijers. *Distributed localization in wireless sensor networks: a quantitative comparison*. The International Journal of Computer and Telecommunication Networking, 43(4):499 – 518, November 2003.
- [9] R. Nagpal, *Organizing a Global Coordinate System from Local Information on an Amorphous Computer*, Massachusetts Institute of Technology, Cambridge, Massachusetts, A.I. MEMO 1666, August 12, 1999.
- [10] K. Pahlavan, L. Xinrong, and J. Makela. *Indoor geolocation science and technology*. IEEE Communications Magazine, 40(2):112–118, February 2002.
- [11] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, *Smart Dust: Communicatng with a Cubic-Millimeter Computer*, vol. 34, pp. 44-51, 2001